US011138038B1

(12) **United States Patent**
Jiang et al.

(10) **Patent No.: US 11,138,038 B1**
(45) **Date of Patent: Oct. 5, 2021**

(54) **ADAPTIVE FREEPOOL SIZE PREDICTION**

(71) Applicant: **Snowflake Inc.**, Bozeman, MT (US)

(72) Inventors: **Qiming Jiang**, Redmond, WA (US); **Orestis Kostakis**, Redmond, WA (US); **Abdul Munir**, San Mateo, CA (US); **Prayag Chandran Nirmala**, Seattle, WA (US); **Jeffrey Rosen**, San Mateo, CA (US)

(73) Assignee: **Snowflake Inc.**, Bozeman, MT (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **17/173,717**

(22) Filed: **Feb. 11, 2021**

Related U.S. Application Data

(60) Provisional application No. 63/143,668, filed on Jan. 29, 2021.

(51) **Int. Cl.**

G06F 9/46	(2006.01)
G06F 9/50	(2006.01)
G06N 5/04	(2006.01)
G06F 16/2455	(2019.01)
G06N 20/00	(2019.01)

(52) **U.S. Cl.**
CPC ***G06F 9/5005*** (2013.01); ***G06F 16/2455*** (2019.01); ***G06N 5/04*** (2013.01); ***G06N 20/00*** (2019.01); *G06F 2209/5011* (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

9,542,296	B1 *	1/2017	Engers	G06F 11/2033
10,067,876	B2 *	9/2018	Bitincka	G06F 12/0802
10,909,114	B1 *	2/2021	Virtuoso	G06F 16/24554
2015/0234682	A1 *	8/2015	Dageville	G06F 16/221 718/104
2018/0196753	A1 *	7/2018	Bitincka	G06F 16/172
2020/0257992	A1 *	8/2020	Achin	G06N 5/04

OTHER PUBLICATIONS

Tang et al., "A Survey on Spark Ecosystem: Big Data Processing Infrastructure, Machine Learning, and Applications", 2019, IEEE, pp. 1-20. (Year: 2019).*

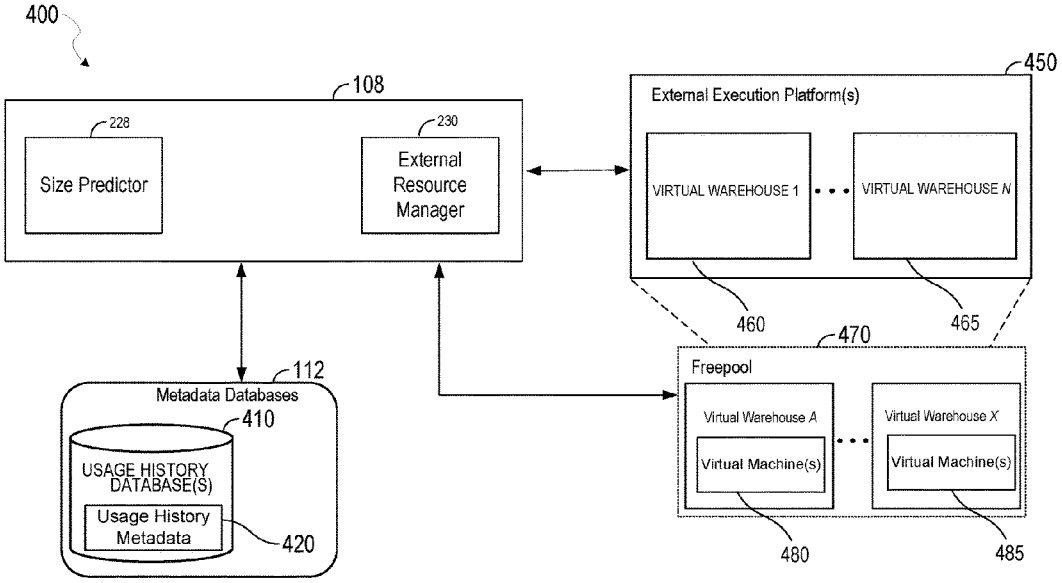
* cited by examiner

Primary Examiner — Kenneth Tang
(74) *Attorney, Agent, or Firm* — Schwegman Lundberg & Woessner, P.A.

(57) **ABSTRACT**

The subject technology determines usage history metadata. The subject technology predicts a size value indicating an amount of computing resources to request for executing a set of queries based at least in part on the usage history metadata. The subject technology determines, during a prefetch window of time within a first period of time, a current size of a freepool of computing resources. The subject technology, in response to the current size of the freepool of computing resources being smaller than the predicted size value, sends a request for additional computing resources to include in the freepool of computing resources. The subject technology receives an indication that the request for additional computing resources was granted. The subject technology performs an operation to include the additional computing resources in the freepool of computing resources.

17 Claims, 6 Drawing Sheets



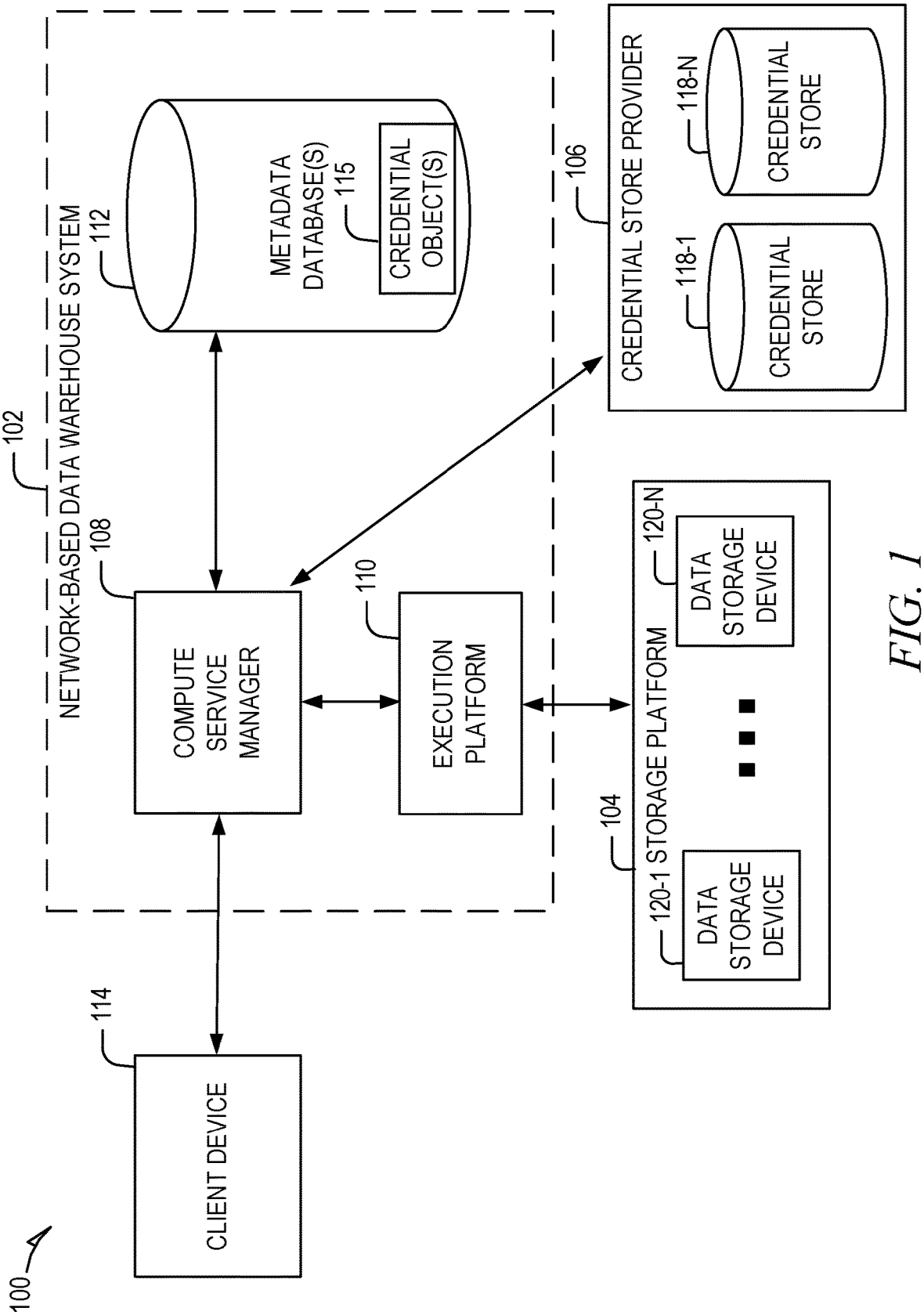

100
CLIENT DEVICE
114
NETWORK-BASED DATA WAREHOUSE SYSTEM
102
COMPUTE SERVICE MANAGER
108
EXECUTION PLATFORM
110
METADATA DATABASE(S)
115
CREDENTIAL OBJECT(S)
112
104
120-1 STORAGE PLATFORM
DATA STORAGE DEVICE
DATA STORAGE DEVICE
120-N
CREDENTIAL STORE PROVIDER
106
CREDENTIAL STORE
118-1
CREDENTIAL STORE
118-N

FIG. 1

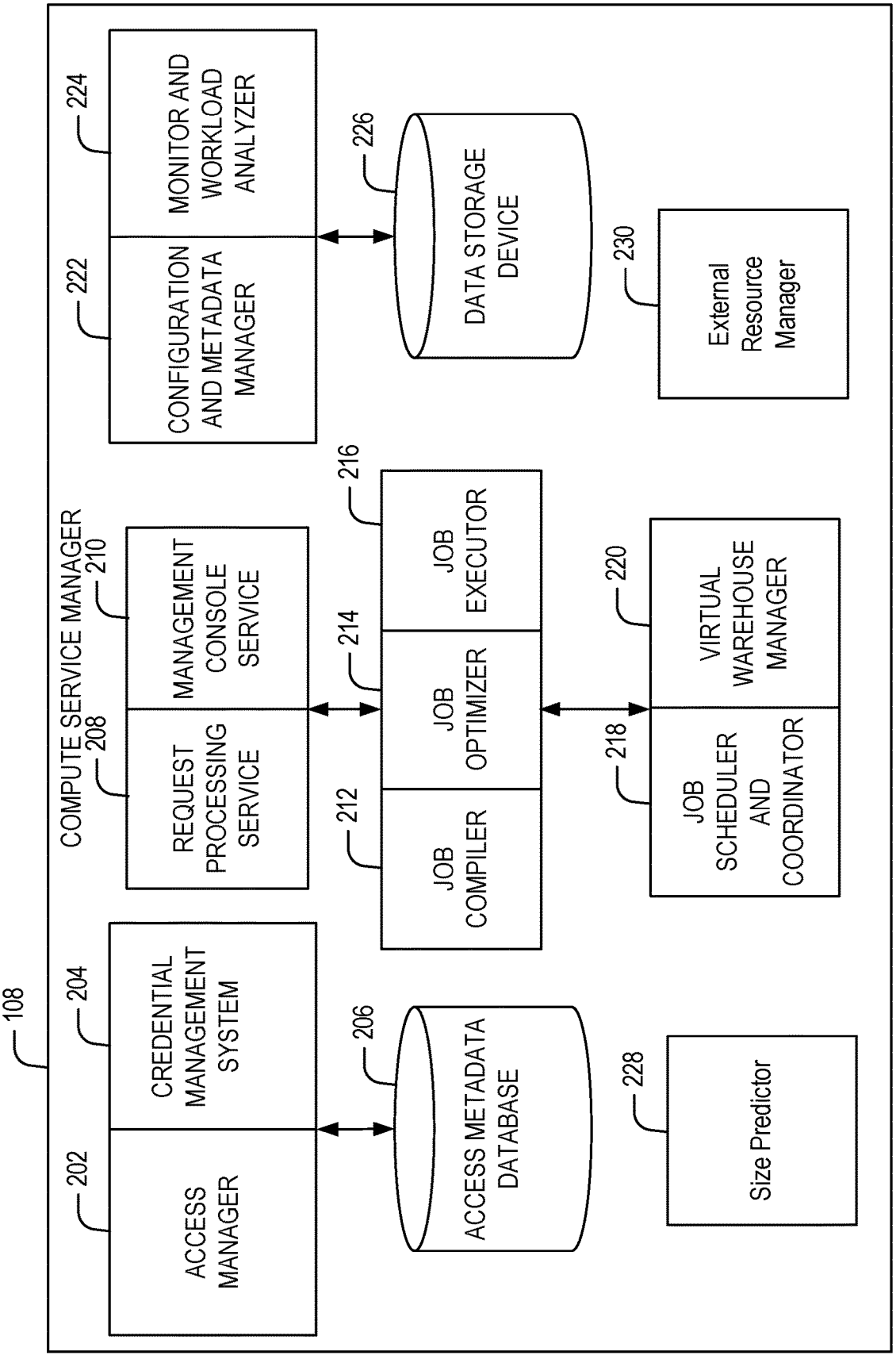

108
COMPUTE SERVICE MANAGER
202
ACCESS MANAGER
204
CREDENTIAL MANAGEMENT SYSTEM
206
ACCESS METADATA DATABASE
208
REQUEST PROCESSING SERVICE
210
MANAGEMENT CONSOLE SERVICE
212
JOB COMPILER
214
JOB OPTIMIZER
216
JOB EXECUTOR
218
JOB SCHEDULER AND COORDINATOR
220
VIRTUAL WAREHOUSE MANAGER
222
CONFIGURATION AND METADATA MANAGER
224
MONITOR AND WORKLOAD ANALYZER
226
DATA STORAGE DEVICE
228
Size Predictor
230
External Resource Manager

FIG. 2

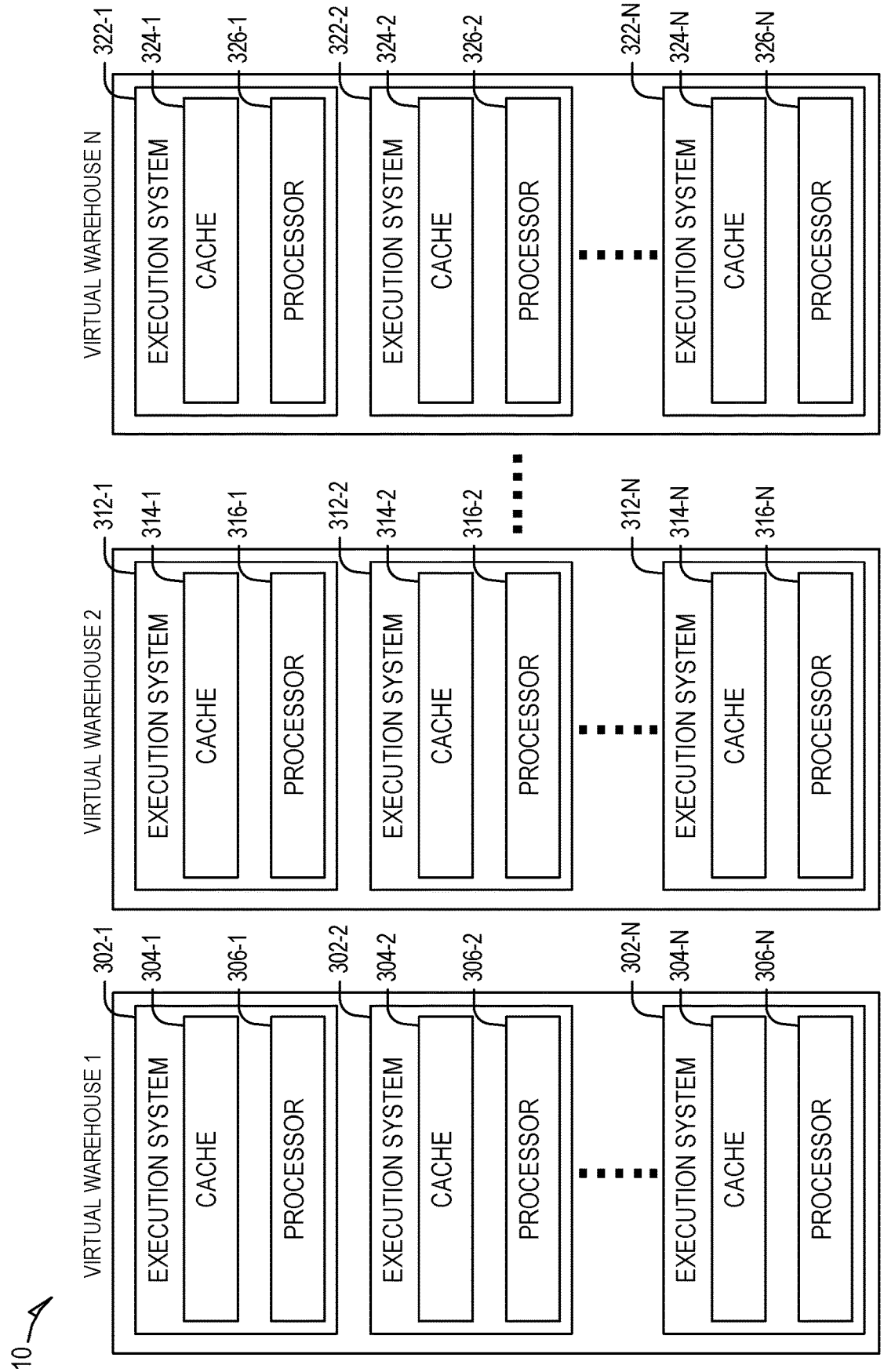

110
VIRTUAL WAREHOUSE 1
VIRTUAL WAREHOUSE 2
VIRTUAL WAREHOUSE N
EXECUTION SYSTEM
CACHE
PROCESSOR
302-1
304-1
306-1
302-2
304-2
306-2
302-N
304-N
306-N
312-1
314-1
316-1
312-2
314-2
316-2
312-N
314-N
316-N
322-1
324-1
326-1
322-2
324-2
326-2
322-N
324-N
326-N

FIG. 3

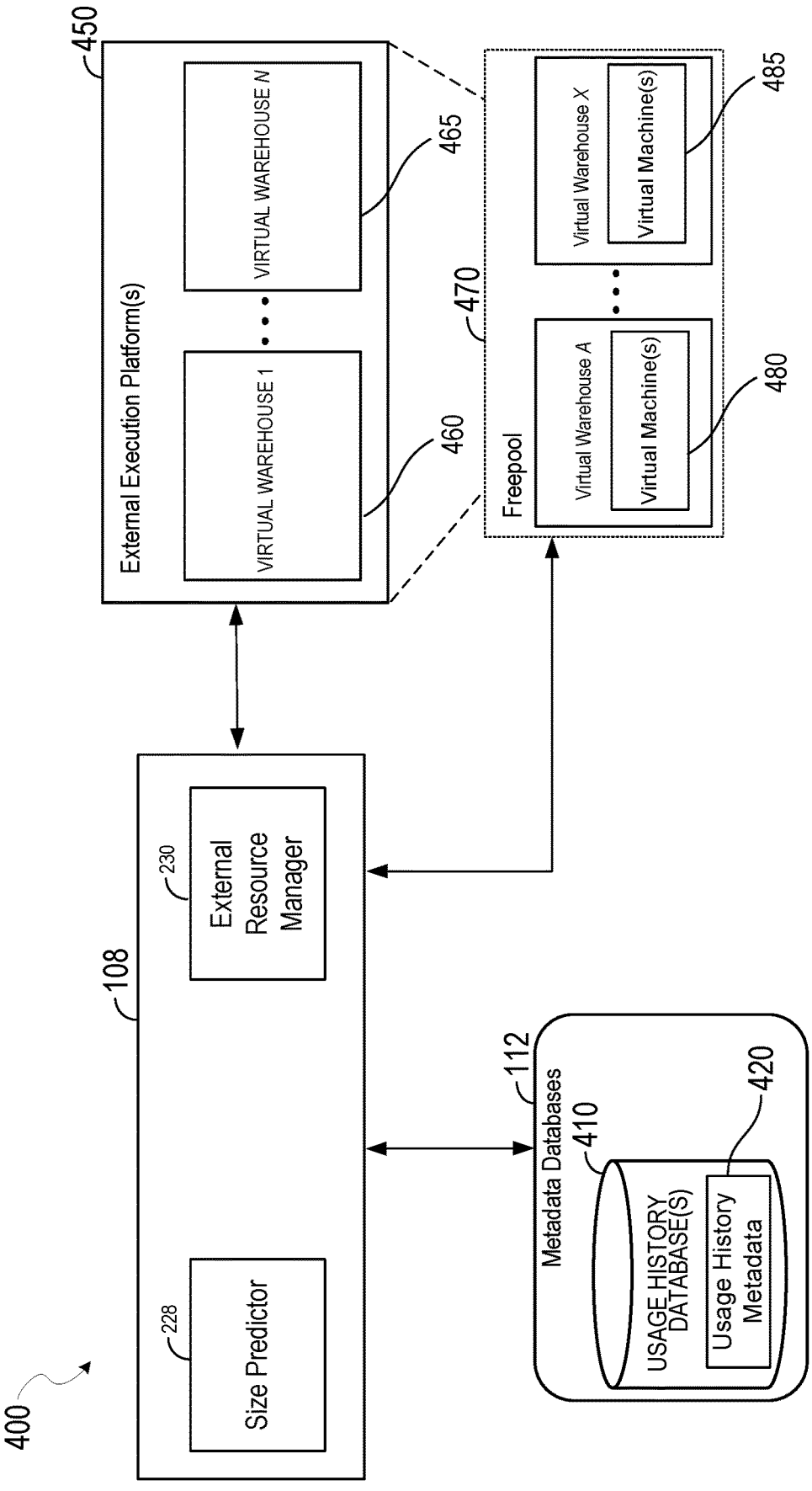

400
450
External Execution Platform(s)
VIRTUAL WAREHOUSE 1
VIRTUAL WAREHOUSE N
460
465
470
Freepool
Virtual Warehouse A
Virtual Machine(s)
Virtual Warehouse X
Virtual Machine(s)
480
485
108
External Resource Manager
230
Size Predictor
228
112
Metadata Databases
410
USAGE HISTORY DATABASE(S)
Usage History Metadata
420

FIG. 4

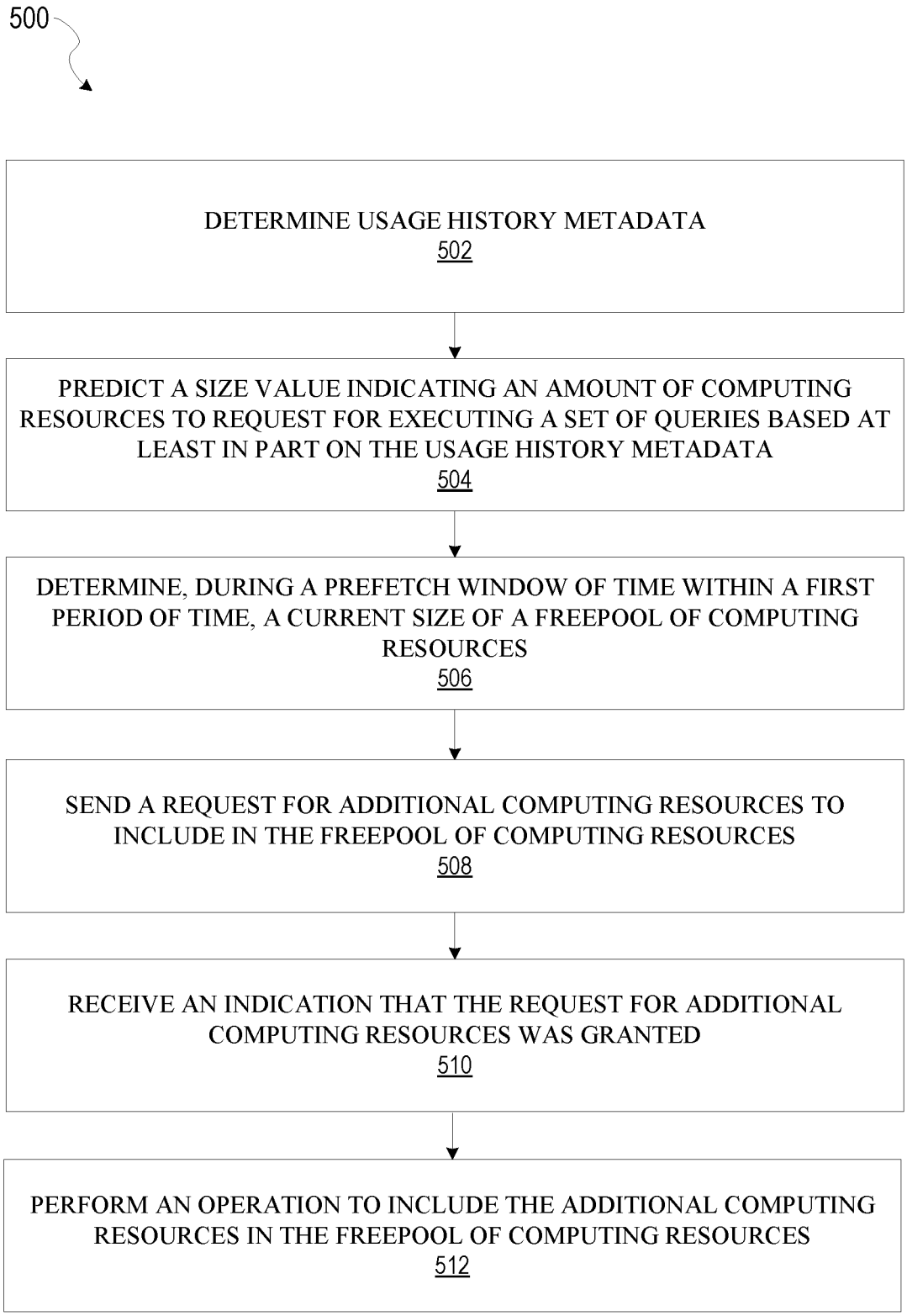

500
DETERMINE USAGE HISTORY METADATA
502
PREDICT A SIZE VALUE INDICATING AN AMOUNT OF COMPUTING RESOURCES TO REQUEST FOR EXECUTING A SET OF QUERIES BASED AT LEAST IN PART ON THE USAGE HISTORY METADATA
504
DETERMINE, DURING A PREFETCH WINDOW OF TIME WITHIN A FIRST PERIOD OF TIME, A CURRENT SIZE OF A FREEPOOL OF COMPUTING RESOURCES
506
SEND A REQUEST FOR ADDITIONAL COMPUTING RESOURCES TO INCLUDE IN THE FREEPOOL OF COMPUTING RESOURCES
508
RECEIVE AN INDICATION THAT THE REQUEST FOR ADDITIONAL COMPUTING RESOURCES WAS GRANTED
510
PERFORM AN OPERATION TO INCLUDE THE ADDITIONAL COMPUTING RESOURCES IN THE FREEPOOL OF COMPUTING RESOURCES
512

FIG. 5

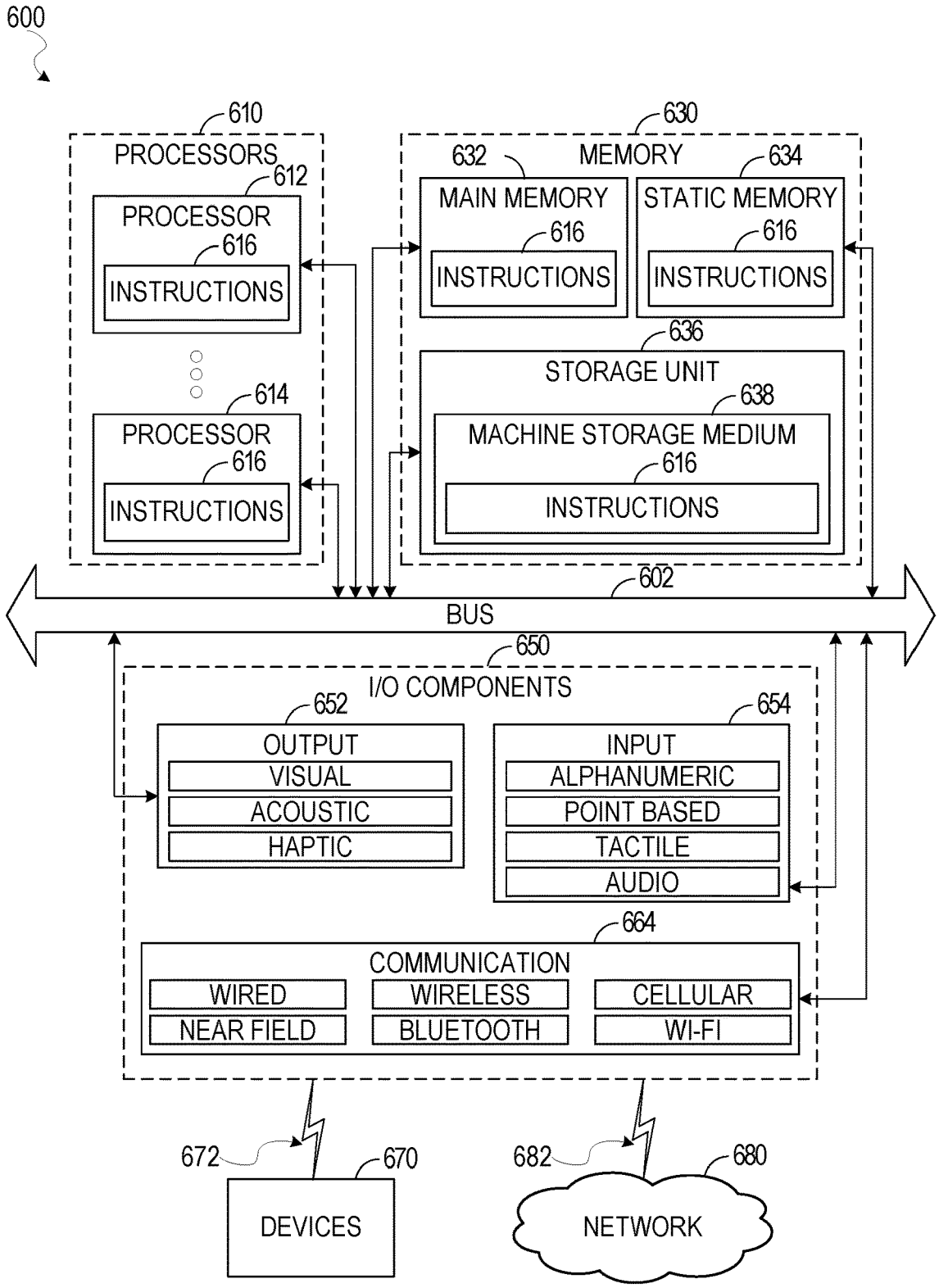

600
610
PROCESSORS
612
PROCESSOR
616
INSTRUCTIONS
614
PROCESSOR
616
INSTRUCTIONS
630
632
MEMORY
634
MAIN MEMORY
616
INSTRUCTIONS
STATIC MEMORY
616
INSTRUCTIONS
636
STORAGE UNIT
638
MACHINE STORAGE MEDIUM
616
INSTRUCTIONS
602
BUS
650
I/O COMPONENTS
652
OUTPUT
VISUAL
ACOUSTIC
HAPTIC
654
INPUT
ALPHANUMERIC
POINT BASED
TACTILE
AUDIO
664
COMMUNICATION
WIRED
WIRELESS
CELLULAR
NEAR FIELD
BLUETOOTH
WI-FI
672
670
DEVICES
682
680
NETWORK

FIG. 6

ADAPTIVE FREEPOOL SIZE PREDICTION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Provisional Patent Application Ser. No. 63/143,668, filed Jan. 29, 2021, entitled "ADAPTIVE FREEPOOL SIZE PREDICTION," and the contents of which is incorporated herein by reference in their entireties for all purposes.

TECHNICAL FIELD

Embodiments of the disclosure relate generally to databases and, more specifically, to enabling performing queries using groups (e.g., pools of computing resources) of servers in an online database system.

BACKGROUND

Databases are widely used for data storage and access in computing applications. A goal of database storage is to provide enormous sums of information in an organized manner so that it can be accessed, managed, and updated. In a database, data may be organized into rows, columns, and tables. Databases are used by various entities and companies for storing information that may need to be accessed or analyzed.

A cloud database is a network-based system used for data analysis and reporting that comprises a central repository of integrated data from one or more disparate sources. A cloud database can store current and historical data that can be used for creating analytical reports for an enterprise based on data stored within databases of the enterprise. To this end, data warehouses typically provide business intelligence tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve metadata.

When certain information is to be extracted from a database, a query statement may be executed against the database data. A cloud database system processes the query and returns certain data according to one or more query predicates that indicate what information should be returned by the query. The data warehouse system extracts specific data from the database and formats that data into a readable form.

BRIEF DESCRIPTION OF THE DRAWINGS

The present disclosure will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the disclosure.

FIG. **1** illustrates an example computing environment that includes a network-based data warehouse system in communication with a cloud storage platform, in accordance with some embodiments of the present disclosure.

FIG. **2** is a block diagram illustrating components of a compute service manager, in accordance with some embodiments of the present disclosure.

FIG. **3** is a block diagram illustrating components of an execution platform, in accordance with some embodiments of the present disclosure.

FIG. **4** is a computing environment conceptually illustrating an example software architecture for providing adaptive freepool size prediction, in accordance with some embodiments of the present disclosure.

FIG. **5** is a flow diagram illustrating operations of a database system in performing a method, in accordance with some embodiments of the present disclosure.

FIG. **6** illustrates a diagrammatic representation of a machine in the form of a computer system within which a set of instructions may be executed for causing the machine to perform any one or more of the methodologies discussed herein, in accordance with some embodiments of the present disclosure.

DETAILED DESCRIPTION

Reference will now be made in detail to specific example embodiments for carrying out the inventive subject matter. Examples of these specific embodiments are illustrated in the accompanying drawings, and specific details are set forth in the following description in order to provide a thorough understanding of the subject matter. It will be understood that these examples are not intended to limit the scope of the claims to the illustrated embodiments. On the contrary, they are intended to cover such alternatives, modifications, and equivalents as may be included within the scope of the disclosure.

Some vendors of cloud database systems may guarantee that users' queries execute on computing resources (e.g., virtual warehouses, servers) within a particular period of time (e.g., "immediately" within a sub-millisecond delay) or according to a set of performance guarantees. Meeting such performance guarantees can be challenging due to virtual warehouses being shutdown (automatically or manually), which now need to be resumed in an instantaneous manner. At the same time, provisioning servers (e.g., virtual machines) from a public cloud provider (PCP) (e.g., AWS, Azure, GCP, and the like), can require waiting in the order of several minutes.

To meet such performance guarantees, the subject system provides a freepool of computing resources (e.g., servers, virtual servers) which are utilized by a given vendor(s) of cloud database systems for executing queries from users. As referred to herein, a "freepool" refers to a group of servers or computing resources (e.g., virtual warehouses) that can be provided by external (e.g., "public" cloud providers) cloud database systems, which are then utilized by the subject system described herein to executing queries.

FIG. **1** illustrates an example computing environment **100** that includes a database system in the example form of a network-based data warehouse system **102**, in accordance with some embodiments of the present disclosure. To avoid obscuring the inventive subject matter with unnecessary detail, various functional components that are not germane to conveying an understanding of the inventive subject matter have been omitted from FIG. **1**. However, a skilled artisan will readily recognize that various additional functional components may be included as part of the computing environment **100** to facilitate additional functionality that is not specifically described herein. In other embodiments, the computing environment may comprise another type of network-based database system or a cloud data platform.

As shown, the computing environment **100** comprises the network-based data warehouse system **102** in communication with a cloud storage platform **104** (e.g., AWS®, Microsoft Azure Blob Storage®, or Google Cloud Storage), and a cloud credential store provider **106**. The network-based data warehouse system **102** is a network-based system used for reporting and analysis of integrated data from one or more disparate sources including one or more storage locations within the cloud storage platform **104**. The cloud

storage platform **104** comprises a plurality of computing machines and provides on-demand computer system resources such as data storage and computing power to the network-based data warehouse system **102**.

The network-based data warehouse system **102** comprises a compute service manager **108**, an execution platform **110**, and one or more metadata databases **112**. The network-based data warehouse system **102** hosts and provides data reporting and analysis services to multiple client accounts.

The compute service manager **108** coordinates and manages operations of the network-based data warehouse system **102**. The compute service manager **108** also performs query optimization and compilation as well as managing clusters of computing services that provide compute resources (also referred to as "virtual warehouses"). The compute service manager **108** can support any number of client accounts such as end users providing data storage and retrieval requests, system administrators managing the systems and methods described herein, and other components/devices that interact with compute service manager **108**.

The compute service manager **108** is also in communication with a client device **114**. The client device **114** corresponds to a user of one of the multiple client accounts supported by the network-based data warehouse system **102**. A user may utilize the client device **114** to submit data storage, retrieval, and analysis requests to the compute service manager **108**.

The compute service manager **108** is also coupled to one or more metadata databases **112** that store metadata pertaining to various functions and aspects associated with the network-based data warehouse system **102** and its users. For example, a metadata database(s) **112** may include a summary of data stored in remote data storage systems as well as data available from a local cache. Additionally, a metadata database(s) **112** may include information regarding how data is organized in remote data storage systems (e.g., the cloud storage platform **104**) and the local caches. Information stored by a metadata database(s) **112** allows systems and services to determine whether a piece of data needs to be accessed without loading or accessing the actual data from a storage device.

As another example, a metadata database(s) **112** can store one or more credential objects **115**. In general, a credential object **115** indicates one or more security credentials to be retrieved from a remote credential store. For example, the credential store provider **106** maintains multiple remote credential stores **118-1** to **118**-N. Each of the remote credential stores **118-1** to **118**-N may be associated with a user account and may be used to store security credentials associated with the user account. A credential object **115** can indicate one of more security credentials to be retrieved by the compute service manager **108** from one of the remote credential stores **118-1** to **118**-N (e.g., for use in accessing data stored by the storage platform **104**).

In an embodiment, a data structure can be utilized for storage of database metadata in the metadata database(s) **112**. For example, such a data structure may be generated from metadata micro-partitions and may be stored in a metadata cache memory. The data structure includes table metadata pertaining to database data stored across a table of the database. The table may include multiple micro-partitions serving as immutable storage devices that cannot be updated in-place. Each of the multiple micro-partitions can include numerous rows and columns making up cells of database data. The table metadata may include a table identification and versioning information indicating, for example, how many versions of the table have been generated over a time period, which version of the table includes the most up-to-date information, how the table was changed over time, and so forth. A new table version may be generated each time a transaction is executed on the table, where the transaction may include a DML, statement such as an insert, delete, merge, and/or update command. Each time a DML statement is executed on the table, and a new table version is generated, one or more new micro-partitions may be generated that reflect the DML statement.

In an embodiment, the aforementioned table metadata includes global information about the table of a specific version. The aforementioned data structure further includes file metadata that includes metadata about a micro-partition of the table. The terms "file" and "micro-partition" may each refer to a subset of database data and may be used interchangeably in some embodiments. The file metadata includes information about a micro-partition of the table. Further, metadata may be stored for each column of each micro-partition of the table. The metadata pertaining to a column of a micro-partition may be referred to as an expression property (EP) and may include any suitable information about the column, including for example, a minimum and maximum for the data stored in the column, a type of data stored in the column, a subject of the data stored in the column, versioning information for the data stored in the column, file statistics for all micro-partitions in the table, global cumulative expressions for columns of the table, and so forth. Each column of each micro-partition of the table may include one or more expression properties.

As mentioned above, a table of a database may include many rows and columns of data. One table may include millions of rows of data and may be very large and difficult to store or read. A very large table may be divided into multiple smaller files corresponding to micro-partitions. For example, one table may be divided into six distinct micro-partitions, and each of the six micro-partitions may include a portion of the data in the table. Dividing the table data into multiple micro-partitions helps to organize the data and to find where certain data is located within the table.

In an embodiment, all data in tables is automatically divided into an immutable storage device referred to as a micro-partition. The micro-partition may be considered a batch unit where each micro-partition has contiguous units of storage. By way of example, each micro-partition may contain between 50 MB and 500 MB of uncompressed data (note that the actual size in storage may be smaller because data may be stored compressed).

Groups of rows in tables may be mapped into individual micro-partitions organized in a columnar fashion. This size and structure allow for extremely granular selection of the micro-partitions to be scanned, which can be comprised of millions, or even hundreds of millions, of micro-partitions. This granular selection process may be referred to herein as "pruning" based on metadata as described further herein.

In an example, pruning involves using metadata to determine which portions of a table, including which micro-partitions or micro-partition groupings in the table, are not pertinent to a query, and then avoiding those non-pertinent micro-partitions when responding to the query and scanning only the pertinent micro-partitions to respond to the query. Metadata may be automatically gathered about all rows stored in a micro-partition, including: the range of values for each of the columns in the micro-partition; the number of distinct values; and/or additional properties used for both optimization and efficient query processing. In one embodiment, micro-partitioning may be automatically performed

on all tables. For example, tables may be transparently partitioned using the ordering that occurs when the data is inserted/loaded.

The micro-partitions as described herein can provide considerable benefits for managing database data, finding database data, and organizing database data. Each micro-partition organizes database data into rows and columns and stores a portion of the data associated with a table. One table may have many micro-partitions. The partitioning of the database data among the many micro-partitions may be done in any manner that makes sense for that type of data.

A query may be executed on a database table to find certain information within the table. To respond to the query, a compute service manager **108** scans the table to find the information requested by the query. The table may include millions and millions of rows, and it would be very time consuming and it would require significant computing resources for the compute service manager **108** to scan the entire table. The micro-partition organization along with the systems, methods, and devices for database metadata storage of the subject technology provide significant benefits by at least shortening the query response time and reducing the amount of computing resources that are required for responding to the query.

The compute service manager **108** may find the cells of database data by scanning database metadata. The multiple level database metadata of the subject technology enable the compute service manager **108** to quickly and efficiently find the correct data to respond to the query. The compute service manager **108** may find the correct table by scanning table metadata across all the multiple tables in a given database. The compute service manager **108** may find a correct grouping of micro-partitions by scanning multiple grouping expression properties across the identified table. Such grouping expression properties include information about database data stored in each of the micro-partitions within the grouping.

The compute service manager **108** may find a correct micro-partition by scanning multiple micro-partition expression properties within the identified grouping of micro-partitions. The compute service manager **108** may find a correct column by scanning one or more column expression properties within the identified micro-partition. The compute service manager **108** may find the correct row(s) by scanning the identified column within the identified micro-partition. The compute service manager **108** may scan the grouping expression properties to find groupings that have data based on the query. The compute service manager **108** reads the micro-partition expression properties for that grouping to find one or more individual micro-partitions based on the query. The compute service manager **108** reads column expression properties within each of the identified individual micro-partitions. The compute service manager **108** scans the identified columns to find the applicable rows based on the query.

In an embodiment, an expression property is information about the one or more columns stored within one or more micro-partitions. For example, multiple expression properties are stored that each pertain to a single column of a single micro-partition. In an alternative embodiment, one or more expression properties are stored that pertain to multiple columns and/or multiple micro-partitions and/or multiple tables. The expression property is any suitable information about the database data and/or the database itself. In an embodiment, the expression property includes one or more of: a summary of database data stored in a column, a type of database data stored in a column, a minimum and maximum for database data stored in a column, a null count for database data stored in a column, a distinct count for database data stored in a column, a structural or architectural indication of how data is stored, and the like.

In an embodiment, the metadata organization structures of the subject technology may be applied to database "pruning" based on the metadata as described further herein. The metadata organization may lead to extremely granular selection of pertinent micro-partitions of a table. Pruning based on metadata is executed to determine which portions of a table of a database include data that is relevant to a query. Pruning is used to determine which micro-partitions or groupings of micro-partitions are relevant to the query, and then scanning only those relevant micro-partitions and avoiding all other non-relevant micro-partitions. By pruning the table based on the metadata, the subject system can save significant time and resources by avoiding all non-relevant micro-partitions when responding to the query. After pruning, the system scans the relevant micro-partitions based on the query.

In an embodiment, the metadata database(s) **112** includes EP files (expression property files), where each of the EP files store a collection of expression properties about corresponding data. Metadata may be stored for each column of each micro-partition of a given table. In an embodiment, the aforementioned EP files can be stored in a cache provided by the subject system for such EP files (e.g., "EP cache").

The compute service manager **108** is further coupled to the execution platform **110**, which provides multiple computing resources that execute various data storage and data retrieval tasks. The execution platform **110** is coupled to storage platform **104** of the cloud storage platform **104**. The storage platform **104** comprises multiple data storage devices **120-1** to **120**-N. In some embodiments, the data storage devices **120-1** to **120**-N are cloud-based storage devices located in one or more geographic locations. For example, the data storage devices **120-1** to **120**-N may be part of a public cloud infrastructure or a private cloud infrastructure. The data storage devices **120-1** to **120**-N may be hard disk drives (HDDs), solid state drives (SSDs), storage clusters, Amazon S3™ storage systems, or any other data storage technology. Additionally, the cloud storage platform **104** may include distributed file systems (such as Hadoop Distributed File Systems (HDFS)), object storage systems, and the like.

The execution platform **110** comprises a plurality of compute nodes. A set of processes on a compute node executes a query plan compiled by the compute service manager **108**. The set of processes can include: a first process to execute the query plan; a second process to monitor and delete cache files using a least recently used (LRU) policy and implement an out of memory (OOM) error mitigation process; a third process that extracts health information from process logs and status to send back to the compute service manager **108**; a fourth process to establish communication with the compute service manager **108** after a system boot; and a fifth process to handle all communication with a compute cluster for a given job provided by the compute service manager **108** and to communicate information back to the compute service manager **108** and other compute nodes of the execution platform **110**.

In some embodiments, communication links between elements of the computing environment **100** are implemented via one or more data communication networks. These data communication networks may utilize any communication protocol and any type of communication medium. In some embodiments, the data communication

networks are a combination of two or more data communication networks (or sub-networks) coupled to one another. In alternate embodiments, these communication links are implemented using any type of communication medium and any communication protocol.

The compute service manager **108**, metadata database(s) **112**, execution platform **110**, and storage platform **104**, are shown in FIG. **1** as individual discrete components. However, each of the compute service manager **108**, metadata database(s) **112**, execution platform **110**, and storage platform **104** may be implemented as a distributed system (e.g., distributed across multiple systems/platforms at multiple geographic locations). Additionally, each of the compute service manager **108**, metadata database(s) **112**, execution platform **110**, and storage platform **104** can be scaled up or down (independently of one another) depending on changes to the requests received and the changing needs of the network-based data warehouse system **102**. Thus, in the described embodiments, the network-based data warehouse system **102** is dynamic and supports regular changes to meet the current data processing needs.

During typical operation, the network-based data warehouse system **102** processes multiple jobs determined by the compute service manager **108**. These jobs are scheduled and managed by the compute service manager **108** to determine when and how to execute the job. For example, the compute service manager **108** may divide the job into multiple discrete tasks and may determine what data is needed to execute each of the multiple discrete tasks. The compute service manager **108** may assign each of the multiple discrete tasks to one or more nodes of the execution platform **110** to process the task. The compute service manager **108** may determine what data is needed to process a task and further determine which nodes within the execution platform **110** are best suited to process the task. Some nodes may have already cached the data needed to process the task and, therefore, be a good candidate for processing the task. Metadata stored in a metadata database(s) **112** assists the compute service manager **108** in determining which nodes in the execution platform **110** have already cached at least a portion of the data needed to process the task. One or more nodes in the execution platform **110** process the task using data cached by the nodes and, if necessary, data retrieved from the cloud storage platform **104**. It is desirable to retrieve as much data as possible from caches within the execution platform **110** because the retrieval speed is typically much faster than retrieving data from the cloud storage platform **104**.

As shown in FIG. **1**, the computing environment **100** separates the execution platform **110** from the storage platform **104**. In this arrangement, the processing resources and cache resources in the execution platform **110** operate independently of the data storage devices **120-1** to **120**-N in the cloud storage platform **104**. Thus, the computing resources and cache resources are not restricted to specific data storage devices **120-1** to **120**-N. Instead, all computing resources and all cache resources may retrieve data from, and store data to, any of the data storage resources in the cloud storage platform **104**.

FIG. **2** is a block diagram illustrating components of the compute service manager **108**, in accordance with some embodiments of the present disclosure. As shown in FIG. **2**, the compute service manager **108** includes an access manager **202** and a credential management system **204** coupled to an access metadata database **206**, which is an example of the metadata database(s) **112**. Access manager **202** handles authentication and authorization tasks for the systems described herein. The credential management system **204** facilitates use of remote stored credentials (e.g., credentials stored in one of the remote credential stores **118-1** to **118**-N) to access external resources such as data resources in a remote storage device. As used herein, the remote storage devices may also be referred to as "persistent storage devices" or "shared storage devices." For example, the credential management system **204** may create and maintain remote credential store definitions and credential objects (e.g., in the access metadata database **206**). A remote credential store definition identifies a remote credential store (e.g., one or more of the remote credential stores **118-1** to **118**-N) and includes access information to access security credentials from the remote credential store. A credential object identifies one or more security credentials using non-sensitive information (e.g., text strings) that are to be retrieved from a remote credential store for use in accessing an external resource. When a request invoking an external resource is received at run time, the credential management system **204** and access manager **202** use information stored in the access metadata database **206** (e.g., a credential object and a credential store definition) to retrieve security credentials used to access the external resource from a remote credential store.

A request processing service **208** manages received data storage requests and data retrieval requests (e.g., jobs to be performed on database data). For example, the request processing service **208** may determine the data to process a received query (e.g., a data storage request or data retrieval request). The data may be stored in a cache within the execution platform **110** or in a data storage device in storage platform **104**.

A management console service **210** supports access to various systems and processes by administrators and other system managers. Additionally, the management console service **210** may receive a request to execute a job and monitor the workload on the system.

The compute service manager **108** also includes a job compiler **212**, a job optimizer **214** and a job executor **216**. The job compiler **212** parses a job into multiple discrete tasks and generates the execution code for each of the multiple discrete tasks. The job optimizer **214** determines the best method to execute the multiple discrete tasks based on the data that needs to be processed. The job executor **216** executes the execution code for jobs received from a queue or determined by the compute service manager **108**.

As further illustrated, the compute service manager **108** includes a size predictor **228**, which is described in further detail in FIG. **4** below. In an example, the size predictor **228** can communicate with an external resource manager **230** as described further herein. In an embodiment, the size predictor **228** generates a prediction for a size of a freepool for execution of an expected set of one or more queries and may return information regarding the same to the external resource manager **230**.

As described further herein, the size predictor **228** can perform operations (e.g., using one or more of a machine learning model, heuristics, rules-based system, and the like) to generate a prediction of computing resources for predicting the size of the freepool based at least in part on analyzing various metadata (e.g., usage history, etc.) received from one or more metadata databases (e.g., metadata database(s) **112**).

A job scheduler and coordinator **218** sends received jobs to the appropriate services or systems for compilation, optimization, and dispatch to the execution platform **110**. For example, jobs may be prioritized and then processed in that prioritized order. In an embodiment, the job scheduler

and coordinator **218** determines a priority for internal jobs that are scheduled by the compute service manager **108** with other "outside" jobs such as user queries that may be scheduled by other systems in the database but may utilize the same processing resources in the execution platform **110**. In some embodiments, the job scheduler and coordinator **218** identifies or assigns particular nodes in the execution platform **110** to process particular tasks. A virtual warehouse manager **220** manages the operation of multiple virtual warehouses implemented in the execution platform **110**. For example, the virtual warehouse manager **220** may generate query plans for executing received queries. Alternatively or conjunctively, the job compiler **212** can generate query plans for executing received queries as discussed further herein.

Additionally, the compute service manager **108** includes a configuration and metadata manager **222**, which manages the information related to the data stored in the remote data storage devices and in the local buffers (e.g., the buffers in execution platform **110**). The configuration and metadata manager **222** uses metadata to determine which data files need to be accessed to retrieve data for processing a particular task or job. A monitor and workload analyzer **224** oversee processes performed by the compute service manager **108** and manages the distribution of tasks (e.g., workload) across the virtual warehouses and execution nodes in the execution platform **110**. The monitor and workload analyzer **224** also redistributes tasks, as needed, based on changing workloads throughout the network-based data warehouse system **102** and may further redistribute tasks based on a user (e.g., "external") query workload that may also be processed by the execution platform **110**. The configuration and metadata manager **222** and the monitor and workload analyzer **224** are coupled to a data storage device **226**. Data storage device **226** in FIG. **2** represents any data storage device within the network-based data warehouse system **102**. For example, data storage device **226** may represent buffers in execution platform **110**, storage devices in storage platform **104**, or any other storage device.

As described in embodiments herein, the compute service manager **108** validates all communication from an execution platform (e.g., the execution platform **110**) to validate that the content and context of that communication are consistent with the task(s) known to be assigned to the execution platform. For example, an instance of the execution platform executing a query A should not be allowed to request access to data-source D (e.g., data storage device **226**) that is not relevant to query A. Similarly, a given execution node (e.g., execution node **302-1** may need to communicate with another execution node (e.g., execution node **302-2**), and should be disallowed from communicating with a third execution node (e.g., execution node **312-1**) and any such illicit communication can be recorded (e.g., in a log or other location). Also, the information stored on a given execution node is restricted to data relevant to the current query and any other data is unusable, rendered so by destruction or encryption where the key is unavailable.

FIG. **3** is a block diagram illustrating components of the execution platform **110**, in accordance with some embodiments of the present disclosure. As shown in FIG. **3**, the execution platform **110** includes multiple virtual warehouses, including virtual warehouse **1**, virtual warehouse **2**, and virtual warehouse n. Each virtual warehouse includes multiple execution nodes that each include a data cache and a processor. The virtual warehouses can execute multiple tasks in parallel by using the multiple execution nodes. As discussed herein, the execution platform **110** can add new virtual warehouses and drop existing virtual warehouses in real-time based on the current processing needs of the systems and users. This flexibility allows the execution platform **110** to quickly deploy large amounts of computing resources when needed without being forced to continue paying for those computing resources when they are no longer needed. All virtual warehouses can access data from any data storage device (e.g., any storage device in cloud storage platform **104**).

Although each virtual warehouse shown in FIG. **3** includes three execution nodes, a particular virtual warehouse may include any number of execution nodes. Further, the number of execution nodes in a virtual warehouse is dynamic, such that new execution nodes are created when additional demand is present, and existing execution nodes are deleted when they are no longer necessary.

Each virtual warehouse is capable of accessing any of the data storage devices **120-1** to **120**-N shown in FIG. **1**. Thus, the virtual warehouses are not necessarily assigned to a specific data storage device **120-1** to **120**-N and, instead, can access data from any of the data storage devices **120-1** to **120**-N within the cloud storage platform **104**. Similarly, each of the execution nodes shown in FIG. **3** can access data from any of the data storage devices **120-1** to **120**-N. In some embodiments, a particular virtual warehouse or a particular execution node may be temporarily assigned to a specific data storage device, but the virtual warehouse or execution node may later access data from any other data storage device.

In the example of FIG. **3**, virtual warehouse **1** includes three execution nodes **302-1**, **302-2**, and **302-*n***. Execution node **302-1** includes a cache **304-1** and a processor **306-1**. Execution node **302-2** includes a cache **304-2** and a processor **306-2**. Execution node **302-*n*** includes a cache **304-*n*** and a processor **306-*n***. Each execution node **302-1**, **302-2**, and **302-*n*** is associated with processing one or more data storage and/or data retrieval tasks. For example, a virtual warehouse may handle data storage and data retrieval tasks associated with an internal service, such as a clustering service, a materialized view refresh service, a file compaction service, a storage procedure service, or a file upgrade service. In other implementations, a particular virtual warehouse may handle data storage and data retrieval tasks associated with a particular data storage system or a particular category of data.

Similar to virtual warehouse **1** discussed above, virtual warehouse **2** includes three execution nodes **312-1**, **312-2**, and **312-*n***. Execution node **312-1** includes a cache **314-1** and a processor **316-1**. Execution node **312-2** includes a cache **314-2** and a processor **316-2**. Execution node **312-*n*** includes a cache **314-*n*** and a processor **316-*n***. Additionally, virtual warehouse **3** includes three execution nodes **322-1**, **322-2**, and **322-*n***. Execution node **322-1** includes a cache **324-1** and a processor **326-1**. Execution node **322-2** includes a cache **324-2** and a processor **326-2**. Execution node **322-*n*** includes a cache **324-*n*** and a processor **326-*n***.

In some embodiments, the execution nodes shown in FIG. **3** are stateless with respect to the data being cached by the execution nodes. For example, these execution nodes do not store or otherwise maintain state information about the execution node or the data being cached by a particular execution node. Thus, in the event of an execution node failure, the failed node can be transparently replaced by another node. Since there is no state information associated with the failed execution node, the new (replacement) execution node can easily replace the failed node without concern for recreating a particular state.

Although the execution nodes shown in FIG. **3** each includes one data cache and one processor, alternate embodiments may include execution nodes containing any number of processors and any number of caches. Additionally, the caches may vary in size among the different execution nodes. The caches shown in FIG. **3** store, in the local execution node, data that was retrieved from one or more data storage devices in cloud storage platform **104**. Thus, the caches reduce or eliminate the bottleneck problems occurring in platforms that consistently retrieve data from remote storage systems. Instead of repeatedly accessing data from the remote storage devices, the systems and methods described herein access data from the caches in the execution nodes, which is significantly faster and avoids the bottleneck problem discussed above. In some embodiments, the caches are implemented using high-speed memory devices that provide fast access to the cached data. Each cache can store data from any of the storage devices in the cloud storage platform **104**.

Further, the cache resources and computing resources may vary between different execution nodes. For example, one execution node may contain significant computing resources and minimal cache resources, making the execution node useful for tasks that require significant computing resources. Another execution node may contain significant cache resources and minimal computing resources, making this execution node useful for tasks that require caching of large amounts of data. Yet another execution node may contain cache resources providing faster input-output operations, useful for tasks that require fast scanning of large amounts of data. In some embodiments, the cache resources and computing resources associated with a particular execution node are determined when the execution node is created, based on the expected tasks to be performed by the execution node.

Additionally, the cache resources and computing resources associated with a particular execution node may change over time based on changing tasks performed by the execution node. For example, an execution node may be assigned more processing resources if the tasks performed by the execution node become more processor-intensive. Similarly, an execution node may be assigned more cache resources if the tasks performed by the execution node require a larger cache capacity.

Although virtual warehouses **1**, **2**, and n are associated with the same execution platform **110**, the virtual warehouses may be implemented using multiple computing systems at multiple geographic locations. For example, virtual warehouse **1** can be implemented by a computing system at a first geographic location, while virtual warehouses **2** and n are implemented by another computing system at a second geographic location. In some embodiments, these different computing systems are cloud-based computing systems maintained by one or more different entities.

Additionally, each virtual warehouse is shown in FIG. **3** as having multiple execution nodes. The multiple execution nodes associated with each virtual warehouse may be implemented using multiple computing systems at multiple geographic locations. For example, an instance of virtual warehouse **1** implements execution nodes **302-1** and **302-2** on one computing platform at a geographic location and implements execution node **302-*n*** at a different computing platform at another geographic location. Selecting particular computing systems to implement an execution node may depend on various factors, such as the level of resources needed for a particular execution node (e.g., processing resource requirements and cache requirements), the resources available at particular computing systems, communication capabilities of networks within a geographic location or between geographic locations, and which computing systems are already implementing other execution nodes in the virtual warehouse.

Execution platform **110** is also fault tolerant. For example, if one virtual warehouse fails, that virtual warehouse is quickly replaced with a different virtual warehouse at a different geographic location.

A particular execution platform **110** may include any number of virtual warehouses. Additionally, the number of virtual warehouses in a particular execution platform is dynamic, such that new virtual warehouses are created when additional processing and/or caching resources are needed. Similarly, existing virtual warehouses may be deleted (e.g., shut down) when the resources associated with the virtual warehouse are no longer necessary.

In some embodiments, the virtual warehouses may operate on the same data in cloud storage platform **104**, but each virtual warehouse has its own execution nodes with independent processing and caching resources. This configuration allows requests on different virtual warehouses to be processed independently and with no interference between the requests. This independent processing, combined with the ability to dynamically add and remove virtual warehouses, supports the addition of new processing capacity for new users without significantly impacting the performance observed by the existing users.

Embodiments of the subject technology enable predicting a size of a freepool in a periodic manner (e.g., for a time interval of a given duration of time). For example, each day is divided into time-intervals of specific duration, which can be referred to as prediction windows. Each prediction window, for example, can be 10 minutes, or some other arbitrary period of time. For each prediction window, the size predictor **228** predicts the optimal size of the freepool. Towards the end of each prediction window, the size predictor **228** prepares for the next prediction window by requesting or releasing enough servers to reach its predicted freepool size. This latter smaller interval of time, within the prediction window, is referred to herein as the "prefetch window".

In an embodiment, predicting the target freepool size value for a specific prediction window is performed by analyzing customer demand during the same time-slots of the last X number of days (e.g., 30 days, but controllable via a parameter). In particular, the subject system analyzes all server-allocation requests (e.g., originating from virtual warehouse create/resume requests) and all server-release (e.g., back to freepool) requests.

FIG. **4** is a computing environment conceptually illustrating an example software architecture for providing adaptive freepool size prediction, which can be performed by components of the compute service manager **108**, in accordance with some embodiments of the present disclosure.

As illustrated, the metadata database(s) **112** includes historical usage data stored as usage history metadata **420** in usage history database(s) **410**, which is also utilized to store any freepool size predictions. To determine the freepool size, the size predictor **228** generates the freepool-size predictions by reading usage history metadata **420**, which is (subsequently) stored in the usage history database(s).

As further shown, the external resource manager **230** is responsible for managing the freepool (e.g., freepool **470**). As illustrated, freepool **470** represents a group of computing resources (e.g., virtual warehouses or servers) that is allocated from external execution platform(s) **450**. The external

execution platform(s) **450** can be provided by an external system such as one or more public cloud providers.

In an embodiment, the external resource manager **230** reads the size prediction values from the usage history database(s) **410** and is responsible for increasing the freepool size, by requesting servers (e.g., in the form of virtual machines (VMs)) from the external execution platform(s) **450**), or decreasing the freepool size, by deleting server(s) (e.g., VMs). The external resource manager **230** also assigns servers (e.g., VMs) to virtual warehouses upon resuming or creating virtual warehouse(s), and for releasing servers back to the freepool **470** when virtual warehouses are shut down.

In an embodiment, the size predictor **228** retrieves usage-history data for the same prediction window of the last X number of days, and determines a target value for the freepool size. Next, the size predictor **228** writes the predicted freepool-size value to the usage history database(s) **410**.

It is appreciated that various techniques can be utilized to predict the target freepool size including machine learning algorithms and models, while other techniques can be based on statistics or statistical models, and the like.

As part of the size predictor **228**, an additional parameter can be included that allows for some 'slack'. For example, a factor can be utilized by which the prediction is multiplied before stored in the usage history database(s) **410**. This allows tuning the freepool size to smaller or larger values, based on our cost-sensitivity or our risk-appetite (since a very large request can always appear out of nowhere).

In an embodiment, the size predictor **228** can compute the prediction sizes for several window sizes (e.g., 6, 10, and 15 minutes). In an example, the prediction window sizes can be based on dividers of the minutes in a day. The values for the prediction window and the prefetch window can be set by a system administrator, and can be set in advance after analyzing a usage-profile of the subject system in some examples.

During the prefetch window of the previous prediction slot, external resource manager **230** reads the predicted freepool size value from the usage history database(s) **410**. The external resource manager **230** checks the current freepool size and compares to the predicted freepool size.

In an embodiment, the external resource manager **230** uses an API provided by external execution platform(s) **450** to request more servers. When the requested servers are available, external resource manager **230** gets new (e.g., requested) servers. Requesting additional servers from external execution platform(s) **450**) is performed asynchronously in an embodiment. Thus, the external resource manager **230** does not fall into a 'lock' state (e.g., waiting for the response) since external the execution platform(s) **450** can take multiple minutes to fulfill the request. External execution platform(s) **450** therefore sends the request, and subsequently performs out other tasks/requests or operations while awaiting the response.

Next, the external resource manager **230** places new servers into the freepool **470**. If the current freepool size is larger than the predicted freepool size, instead of requesting additional servers, external resource manager **230** returns the necessary amount of servers back to the external execution platform(s) **450** (e.g., shut down the VMs).

In an embodiment, when the size predictor **228** fails or is unavailable, another component of the compute service manager **108** can perform the prediction just in time the prediction is not already stored in the metadata database(s) **112**.

Moreover, although the above examples may be related to a single freepool, it is appreciated that multiple freepools can be utilized simultaneously and independently where each freepool can include different types of VMs.

Public cloud providers (PCP), such as the external execution platform(s) **450**, are complex systems and can experience service degradation of their own. As a result, the PCP's latency (response time) to any requests from the external resource manager **230** requesting additional servers fluctuates over time. The aloof solution is to do nothing. Alternatively, the external resource manager **230** can monitor the response time of the PCP and if that increases beyond an acceptable threshold for a specific prefetch-window size, then increase the prefetch window size and the prediction window size.

This is achievable by having the size predictor **228** compute the prediction size for slots of multiple sizes (i.e. different amount of minutes), and if the PCP is slow enough, then the external resource manager **230** can fall back to using larger prefetch and prediction window sizes.

FIG. **5** is a flow diagram illustrating operations of a database system in performing a method, in accordance with some embodiments of the present disclosure. The method **500** may be embodied in computer-readable instructions for execution by one or more hardware components (e.g., one or more processors) such that the operations of the method **500** may be performed by components of network-based data warehouse system **102**, such as components of the compute service manager **108**. Accordingly, the method **500** is described below, by way of example with reference thereto. However, it shall be appreciated that the method **500** may be deployed on various other hardware configurations and is not intended to be limited to deployment within the network-based data warehouse system **102**.

At operation **502**, the size predictor **228** determines usage history metadata. At operation **504**, the size predictor **228** predicts a size value indicating an amount of computing resources to request for executing a set of queries based at least in part on the usage history metadata. At operation **506**, the size predictor **228** determines, during a prefetch window of time within a first period of time, a current size of a freepool of computing resources. At operation **508**, in response to the current size of the freepool of computing resources being smaller than the predicted size value, the external resource manager **230** sends a request for additional computing resources to include in the freepool of computing resources. At operation **510**, the external resource manager **230** receives an indication that the request for additional computing resources was granted. At operation **512**, the external resource manager **230** performs an operation to include the additional computing resources in the freepool of computing resources.

FIG. **6** illustrates a diagrammatic representation of a machine **600** in the form of a computer system within which a set of instructions may be executed for causing the machine **600** to perform any one or more of the methodologies discussed herein, according to an example embodiment. Specifically, FIG. **6** shows a diagrammatic representation of the machine **600** in the example form of a computer system, within which instructions **616** (e.g., software, a program, an application, an applet, an app, or other executable code) for causing the machine **600** to perform any one or more of the methodologies discussed herein may be executed. For example, the instructions **616** may cause the machine **600** to execute any one or more operations of the method **500**. As another example, the instructions **616** may cause the machine **600** to implement portions of the data

flows illustrated in at least FIG. **4**. In this way, the instructions **616** transform a general, non-programmed machine into a particular machine **600** (e.g., the compute service manager **108** or a node in the execution platform **110**) that is specially configured to carry out any one of the described and illustrated functions in the manner described herein.

In alternative embodiments, the machine **600** operates as a standalone device or may be coupled (e.g., networked) to other machines. In a networked deployment, the machine **600** may operate in the capacity of a server machine or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine **600** may comprise, but not be limited to, a server computer, a client computer, a personal computer (PC), a tablet computer, a laptop computer, a netbook, a smart phone, a mobile device, a network router, a network switch, a network bridge, or any machine capable of executing the instructions **616**, sequentially or otherwise, that specify actions to be taken by the machine **600**. Further, while only a single machine **600** is illustrated, the term "machine" shall also be taken to include a collection of machines **600** that individually or jointly execute the instructions **616** to perform any one or more of the methodologies discussed herein.

The machine **600** includes processors **610**, memory **630**, and input/output (I/O) components **650** configured to communicate with each other such as via a bus **602**. In an example embodiment, the processors **610** (e.g., a central processing unit (CPU), a reduced instruction set computing (RISC) processor, a complex instruction set computing (CISC) processor, a graphics processing unit (GPU), a digital signal processor (DSP), an application-specific integrated circuit (ASIC), a radio-frequency integrated circuit (RFIC), another processor, or any suitable combination thereof) may include, for example, a processor **612** and a processor **614** that may execute the instructions **616**. The term "processor" is intended to include multi-core processors **610** that may comprise two or more independent processors (sometimes referred to as "cores") that may execute instructions **616** contemporaneously. Although FIG. **6** shows multiple processors **610**, the machine **600** may include a single processor with a single core, a single processor with multiple cores (e.g., a multi-core processor), multiple processors with a single core, multiple processors with multiple cores, or any combination thereof.

The memory **630** may include a main memory **632**, a static memory **634**, and a storage unit **636**, all accessible to the processors **610** such as via the bus **602**. The main memory **632**, the static memory **634**, and the storage unit **636** store the instructions **616** embodying any one or more of the methodologies or functions described herein. The instructions **616** may also reside, completely or partially, within the main memory **632**, within the static memory **634**, within machine storage medium **638** of the storage unit **636**, within at least one of the processors **610** (e.g., within the processor's cache memory), or any suitable combination thereof, during execution thereof by the machine **600**.

The I/O components **650** include components to receive input, provide output, produce output, transmit information, exchange information, capture measurements, and so on. The specific I/O components **650** that are included in a particular machine **600** will depend on the type of machine. For example, portable machines such as mobile phones will likely include a touch input device or other such input mechanisms, while a headless server machine will likely not include such a touch input device. It will be appreciated that the I/O components **650** may include many other components that are not shown in FIG. **6**. The I/O components **650** are grouped according to functionality merely for simplifying the following discussion and the grouping is in no way limiting. In various example embodiments, the I/O components **650** may include output components **652** and input components **654**. The output components **652** may include visual components (e.g., a display such as a plasma display panel (PDP), a light emitting diode (LED) display, a liquid crystal display (LCD), a projector, or a cathode ray tube (CRT)), acoustic components (e.g., speakers), other signal generators, and so forth. The input components **654** may include alphanumeric input components (e.g., a keyboard, a touch screen configured to receive alphanumeric input, a photo-optical keyboard, or other alphanumeric input components), point-based input components (e.g., a mouse, a touchpad, a trackball, a joystick, a motion sensor, or another pointing instrument), tactile input components (e.g., a physical button, a touch screen that provides location and/or force of touches or touch gestures, or other tactile input components), audio input components (e.g., a microphone), and the like.

Communication may be implemented using a wide variety of technologies. The I/O components **650** may include communication components **664** operable to couple the machine **600** to a network **680** or devices **670** via a coupling **682** and a coupling **672**, respectively. For example, the communication components **664** may include a network interface component or another suitable device to interface with the network **680**. In further examples, the communication components **664** may include wired communication components, wireless communication components, cellular communication components, and other communication components to provide communication via other modalities. The devices **670** may be another machine or any of a wide variety of peripheral devices (e.g., a peripheral device coupled via a universal serial bus (USB)). For example, as noted above, the machine **600** may correspond to any one of the compute service manager **108** or the execution platform **110**, and the devices **670** may include the client device **114** or any other computing device described herein as being in communication with the network-based data warehouse system **102** or the cloud storage platform **104**.

Executable Instructions and Machine Storage Medium

The various memories (e.g., **630**, **632**, **634**, and/or memory of the processor(s) **610** and/or the storage unit **636**) may store one or more sets of instructions **616** and data structures (e.g., software) embodying or utilized by any one or more of the methodologies or functions described herein. These instructions **616**, when executed by the processor(s) **610**, cause various operations to implement the disclosed embodiments.

As used herein, the terms "machine-storage medium," "device-storage medium," and "computer-storage medium" mean the same thing and may be used interchangeably in this disclosure. The terms refer to a single or multiple storage devices and/or media (e.g., a centralized or distributed database, and/or associated caches and servers) that store executable instructions and/or data. The terms shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media, including memory internal or external to processors. Specific examples of machine-storage media, computer-storage media, and/or device-storage media include non-volatile memory, including by way of example semiconductor memory devices, e.g., erasable programmable read-only memory (EPROM), electrically erasable programmable read-only memory (EEPROM), field-programmable gate

arrays (FPGAs), and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The terms "machine-storage media," "computer-storage media," and "device-storage media" specifically exclude carrier waves, modulated data signals, and other such media, at least some of which are covered under the term "signal medium" discussed below.

Transmission Medium

In various example embodiments, one or more portions of the network **680** may be an ad hoc network, an intranet, an extranet, a virtual private network (VPN), a local-area network (LAN), a wireless LAN (WLAN), a wide-area network (WAN), a wireless WAN (WWAN), a metropolitan-area network (MAN), the Internet, a portion of the Internet, a portion of the public switched telephone network (PSTN), a plain old telephone service (POTS) network, a cellular telephone network, a wireless network, a Wi-Fi® network, another type of network, or a combination of two or more such networks. For example, the network **680** or a portion of the network **680** may include a wireless or cellular network, and the coupling **682** may be a Code Division Multiple Access (CDMA) connection, a Global System for Mobile communications (GSM) connection, or another type of cellular or wireless coupling. In this example, the coupling **682** may implement any of a variety of types of data transfer technology, such as Single Carrier Radio Transmission Technology (1×RTT), Evolution-Data Optimized (EVDO) technology, General Packet Radio Service (GPRS) technology, Enhanced Data rates for GSM Evolution (EDGE) technology, third Generation Partnership Project (3GPP) including 3G, fourth generation wireless (4G) networks, Universal Mobile Telecommunications System (UMTS), High-Speed Packet Access (HSPA), Worldwide Interoperability for Microwave Access (WiMAX), Long Term Evolution (LTE) standard, others defined by various standard-setting organizations, other long-range protocols, or other data transfer technology.

The instructions **616** may be transmitted or received over the network **680** using a transmission medium via a network interface device (e.g., a network interface component included in the communication components **664**) and utilizing any one of a number of well-known transfer protocols (e.g., hypertext transfer protocol (HTTP)). Similarly, the instructions **616** may be transmitted or received using a transmission medium via the coupling **672** (e.g., a peer-to-peer coupling) to the devices **670**. The terms "transmission medium" and "signal medium" mean the same thing and may be used interchangeably in this disclosure. The terms "transmission medium" and "signal medium" shall be taken to include any intangible medium that is capable of storing, encoding, or carrying the instructions **616** for execution by the machine **600**, and include digital or analog communications signals or other intangible media to facilitate communication of such software. Hence, the terms "transmission medium" and "signal medium" shall be taken to include any form of modulated data signal, carrier wave, and so forth. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal.

Computer-Readable Medium

The terms "machine-readable medium," "computer-readable medium," and "device-readable medium" mean the same thing and may be used interchangeably in this disclosure. The terms are defined to include both machine-storage media and transmission media. Thus, the terms include both storage devices/media and carrier waves/modulated data signals.

The various operations of example methods described herein may be performed, at least partially, by one or more processors that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Similarly, the methods described herein may be at least partially processor-implemented. For example, at least some of the operations of the method **500** may be performed by one or more processors. The performance of certain of the operations may be distributed among the one or more processors, not only residing within a single machine, but also deployed across a number of machines. In some example embodiments, the processor or processors may be located in a single location (e.g., within a home environment, an office environment, or a server farm), while in other embodiments the processors may be distributed across a number of locations.

Conclusion

Although the embodiments of the present disclosure have been described with reference to specific example embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader scope of the inventive subject matter. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense. The accompanying drawings that form a part hereof show, by way of illustration, and not of limitation, specific embodiments in which the subject matter may be practiced. The embodiments illustrated are described in sufficient detail to enable those skilled in the art to practice the teachings disclosed herein. Other embodiments may be used and derived therefrom, such that structural and logical substitutions and changes may be made without departing from the scope of this disclosure. This Detailed Description, therefore, is not to be taken in a limiting sense, and the scope of various embodiments is defined only by the appended claims, along with the full range of equivalents to which such claims are entitled.

Such embodiments of the inventive subject matter may be referred to herein, individually and/or collectively, by the term "invention" merely for convenience and without intending to voluntarily limit the scope of this application to any single invention or inventive concept if more than one is in fact disclosed. Thus, although specific embodiments have been illustrated and described herein, it should be appreciated that any arrangement calculated to achieve the same purpose may be substituted for the specific embodiments shown. This disclosure is intended to cover any and all adaptations or variations of various embodiments. Combinations of the above embodiments, and other embodiments not specifically described herein, will be apparent, to those of skill in the art, upon reviewing the above description.

In this document, the terms "a" or "an" are used, as is common in patent documents, to include one or more than one, independent of any other instances or usages of "at least one" or "one or more." In this document, the term "or" is used to refer to a nonexclusive or, such that "A or B" includes "A but not B," "B but not A," and "A and B," unless otherwise indicated. In the appended claims, the terms "including" and "in which" are used as the plain-English equivalents of the respective terms "comprising" and "wherein." Also, in the following claims, the terms "including" and "comprising" are open-ended; that is, a system, device, article, or process that includes elements in addition

to those listed after such a term in a claim is still deemed to fall within the scope of that claim.

What is claimed is:

1. A system comprising:

at least one hardware processor; and

a memory storing instructions that cause the at least one hardware processor to perform operations comprising:

determining usage history metadata, the determining the usage history metadata comprising:

requesting information related to the usage history metadata from a metadata database, and

receiving the requested information from the metadata database, the requested information comprising information related to user demand during corresponding time slots of a number of prior days, the information related to user demand comprising requests for allocation of additional computing resources to a freepool of computing resources and requests for releasing computing resources from the freepool of computing resources;

predicting a size value indicating an amount of computing resources to request for executing a set of queries based at least in part on the usage history metadata;

determining, during a prefetch window of time within a first period of time, a current size of the freepool of computing resources;

in response to the current size of the freepool of computing resources being smaller than the predicted size value, sending a request for additional computing resources to include in the freepool of computing resources;

receiving an indication that the request for additional computing resources was granted; and

performing an operation to include the additional computing resources in the freepool of computing resources.

2. The system of claim **1**, wherein the operations further comprise:

storing the predicted size value in a metadata database; and

reading, prior to predicting the size value, the stored predicted size value from the metadata database.

3. The system of claim **2**, wherein the operations further comprise:

in response to the current size of the freepool of computing resources being greater than the predicted size value, sending a request for removing computing resources from the freepool of computing resources.

4. The system of claim **1**, wherein predicting the size value indicating the amount of computing resources to request for executing the set of queries is performed based on a machine learning model.

5. The system of claim **1**, wherein predicting the size value indicating the amount of computing resources to request for executing the set of queries is performed based on a set of statistics or a statistical model.

6. The system of claim **1**, wherein the predicted size value is multiplied by a factor prior to storing in a metadata database.

7. The system of claim **1**, wherein predicting the size value and sending the request for additional computing resources are performed by different components of a database system.

8. The system of claim **1**, wherein another database operation is performed after sending the request for additional computing resources and before receiving the indication that the request for additional computing resources was granted.

9. A method comprising:

determining usage history metadata, the determining the usage history metadata comprising:

requesting information related to the usage history metadata from a metadata database, and

receiving the requested information from the metadata database, the requested information comprising information related to user demand during corresponding time slots of a number of prior days, the information related to user demand comprising requests for allocation of additional computing resources to a freepool of computing resources and requests for releasing computing resources from the freepool of computing resources;

predicting a size value indicating an amount of computing resources to request for executing a set of queries based at least in part on the usage history metadata;

determining, during a prefetch window of time within a first period of time, a current size of the freepool of computing resources;

in response to the current size of the freepool of computing resources being smaller than the predicted size value, sending a request for additional computing resources to include in the freepool of computing resources;

receiving an indication that the request for additional computing resources was granted; and

performing an operation to include the additional computing resources in the freepool of computing resources.

10. The method of claim **9**, further comprising:

storing the predicted size value in a metadata database; and

reading, prior to predicting the size value, the stored predicted size value from the metadata database.

11. The method of claim **10**, further comprising:

in response to the current size of the freepool of computing resources being greater than the predicted size value, sending a request for removing computing resources from the freepool of computing resources.

12. The method of claim **9**, wherein predicting the size value indicating the amount of computing resources to request for executing the set of queries is performed based on a machine learning model.

13. The method of claim **9**, wherein predicting the size value indicating the amount of computing resources to request for executing the set of queries is performed based on a set of statistics or a statistical model.

14. The method of claim **9**, wherein the predicted size value is multiplied by a factor prior to storing in a metadata database.

15. The method of claim **9**, wherein predicting the size value and sending the request for additional computing resources are performed by different components of a database system.

16. The method of claim **9**, wherein another database operation is performed after sending the request for additional computing resources and before receiving the indication that the request for additional computing resources was granted.

17. A non-transitory computer-storage medium comprising instructions that, when executed by one or more processors of a machine, configure the machine to perform operations comprising:

determining usage history metadata, the determining the usage history metadata comprising:

requesting information related to the usage history metadata from a metadata database, and

receiving the requested information from the metadata database, the requested information comprising information related to user demand during corresponding time slots of a number of prior days, the information related to user demand comprising requests for allocation of additional computing resources to a freepool of computing resources and requests for releasing computing resources from the freepool of computing resources;

predicting a size value indicating an amount of computing resources to request for executing a set of queries based at least in part on the usage history metadata;

determining, during a prefetch window of time within a first period of time, a current size of the freepool of computing resources;

in response to the current size of the freepool of computing resources being smaller than the predicted size value, sending a request for additional computing resources to include in the freepool of computing resources;

receiving an indication that the request for additional computing resources was granted; and

performing an operation to include the additional computing resources in the freepool of computing resources.

* * * * *